

ATLAS
MANUAL DE USUARIO DEL ARQUETIPO WEBSERVICE

Versión 1.8

Área de Aplicaciones Especiales y Arquitectura de Software



Agencia de Informática y Comunicaciones de la Comunidad de Madrid



Hoja de Control

Título	Manual de usuario del Arquetipo WebService		
Documento de Referencia	NORMATIVA ATLAS		
Responsable	Área de Aplicaciones Especiales y Arquitectura de Software		
Versión	1.8	**Fecha Versión**	01/03/2013

Registro de Cambios

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.0	Versión inicial del documento	Área de Integración y Arquitectura de Aplicaciones	25/05/2010
1.1	- Nueva configuración de Arquetipo Web Service - Incluida creación previa del arquetipo de proyecto - Nueva nomenclatura de paquetes	Área de Integración y Arquitectura de Aplicaciones	16/09/2010
1.2	- Modificaciones para versión 1.1.0 de ATLAS	Área de Integración y Arquitectura de Aplicaciones	16/02/2011
1.3	- Las preguntas frecuentes se consultarán en el portal de arquitectura. Se modificad el nombre del área	Área de Aplicaciones Especiales y Arquitectura de Software	05/07/2011
1.4	- Nueva arquitectura modular de arquetipo de servicio web	Área de Aplicaciones Especiales y Arquitectura de Software	18/10/2011
1.5	- Nuevo módulo test con proyectos de SoapUI 4.0.1	Área de Aplicaciones Especiales y Arquitectura de Software	21/11/2011
1.6	- Se añaden las clases BaseService y BaseServiceImpl, así como los servicios y DAOs para EstadoCivil - Se añade a la estructura del módulo web la carpeta “generador”. - Se añaden las clases de test agrupadas por servicio	Área de Aplicaciones Especiales y Arquitectura de Software	24/02/2012
1.7	Se elimina las clases para la fachada ya que en los servicios web no tiene sentido. Internamente se ha modificado la versión de axis por lo tanto esta versión del arquetipo utilizada esta nueva versión.	Área de Aplicaciones Especiales y Arquitectura de Software	29/10/2012

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.8	Revisión completa del documento por nueva estructura de arquetipo en ATLAS 1.2.3	Área de Aplicaciones Especiales y Arquitectura de Software	27/02/2013

Índice

1 INTRODUCCIÓN

Los arquetipos son las plantillas para la generación de los distintos proyectos dentro del Framework Atlas. Estos arquetipos utilizan el plugin archetype de maven para generar la estructura de ficheros y directorios necesarios para nuestro proyecto, gestionando las librerías que le indiquemos así como las dependencias. Todas las librerías serán incluidas durante el empaquetado del proyecto, por lo que para generar y compilar un arquetipo debe estar conectado al repositorio de artefactos de la Comunidad de Madrid (Artifactory). El framework Atlas consta de los siguientes arquetipos:

- Arquetipo para inicio de proyecto
- Arquetipo para módulos de tipo web
- Arquetipo para módulos de tipo jar.
- Arquetipo para módulos de tipo webservice o servicios web
- Arquetipo para módulos de tipo batch
- Arquetipo para módulos de gestión documental (documentum)

El **Arquetipo WebService** genera un proyecto modular con un módulo maven de tipo war para el servicio web listo para ser desplegado, y un módulo de tipo jar para el cliente de dicho servicio. Está preparado para publicar un servicio de Spring a través de Axis2 sin generación de código intermedio. El proyecto viene preconfigurado para acceder a base de datos a través de Hibernate y publica como webservices una serie de servicios de ejemplo.

Se incluye un ejemplo de servicio sin seguridad, otro seguro a nivel de transporte y otro securizado con webservice security. Para más información sobre estos tipos de seguridad consultar el manual ATLAS_MUS_Servicios_Web.

1.1 AUDIENCIA OBJETIVO

Este documento está dirigido a desarrolladores de proyectos java para ICM en los que se desee crear una aplicación que publique un Servicio Web utilizando el framework de desarrollo Atlas.

1.2 CONOCIMIENTOS PREVIOS

Para un completo entendimiento del documento, el lector deberá tener conocimientos previos sobre las siguientes tecnologías:

- Java
- Eclipse
- Maven
- Spring Framework.
- Axis2 y SoapUI

Además, es necesario haber leído y seguido los pasos del manual **ATLAS_MUS_Preparacion_Entorno_Desarrollo** para tener el entorno de desarrollo preparado.

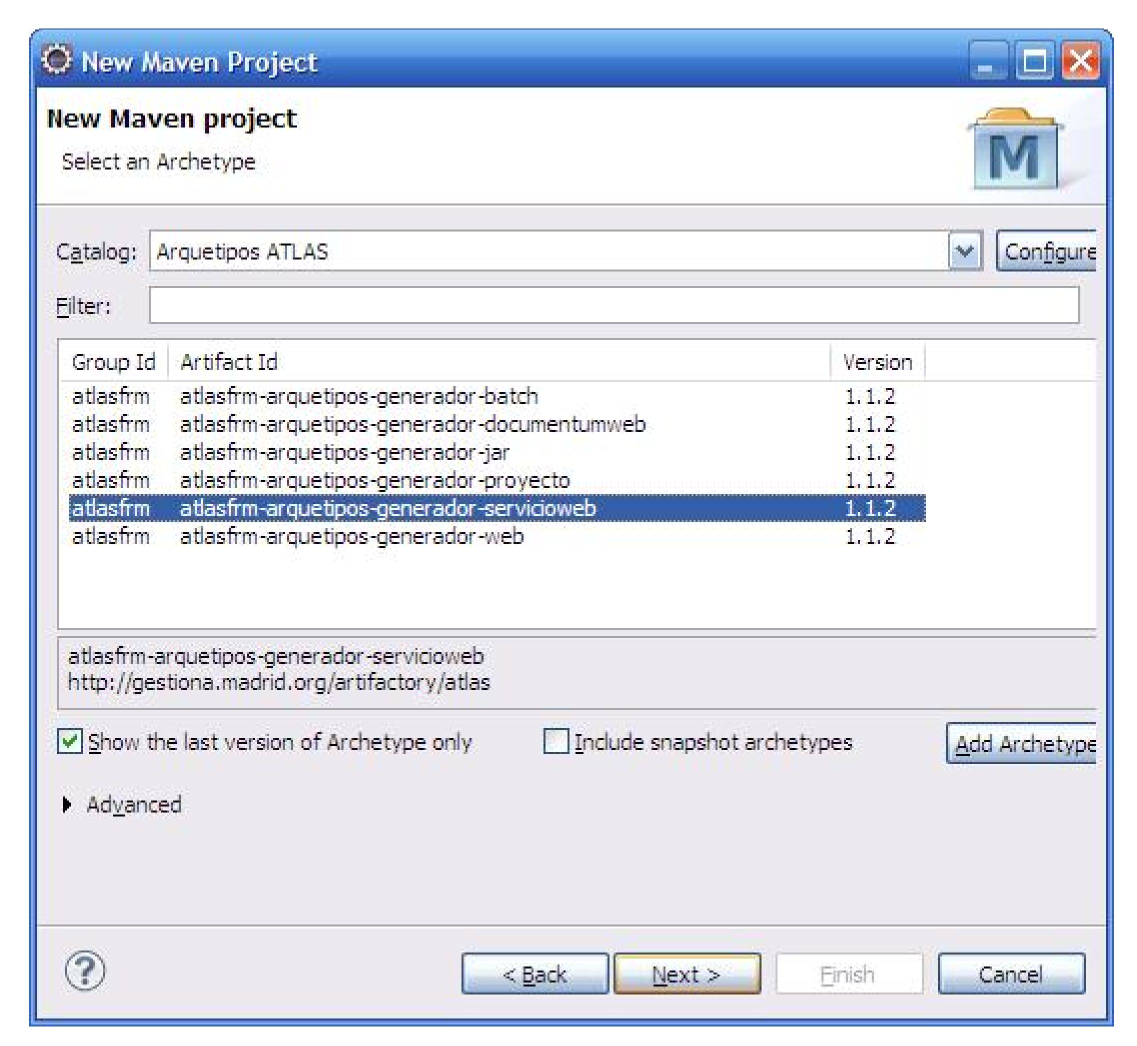
2 INFORMACIÓN SOBRE EL ARQUETIPO

2.1 Creación de una aplicación partiendo del arquetipo

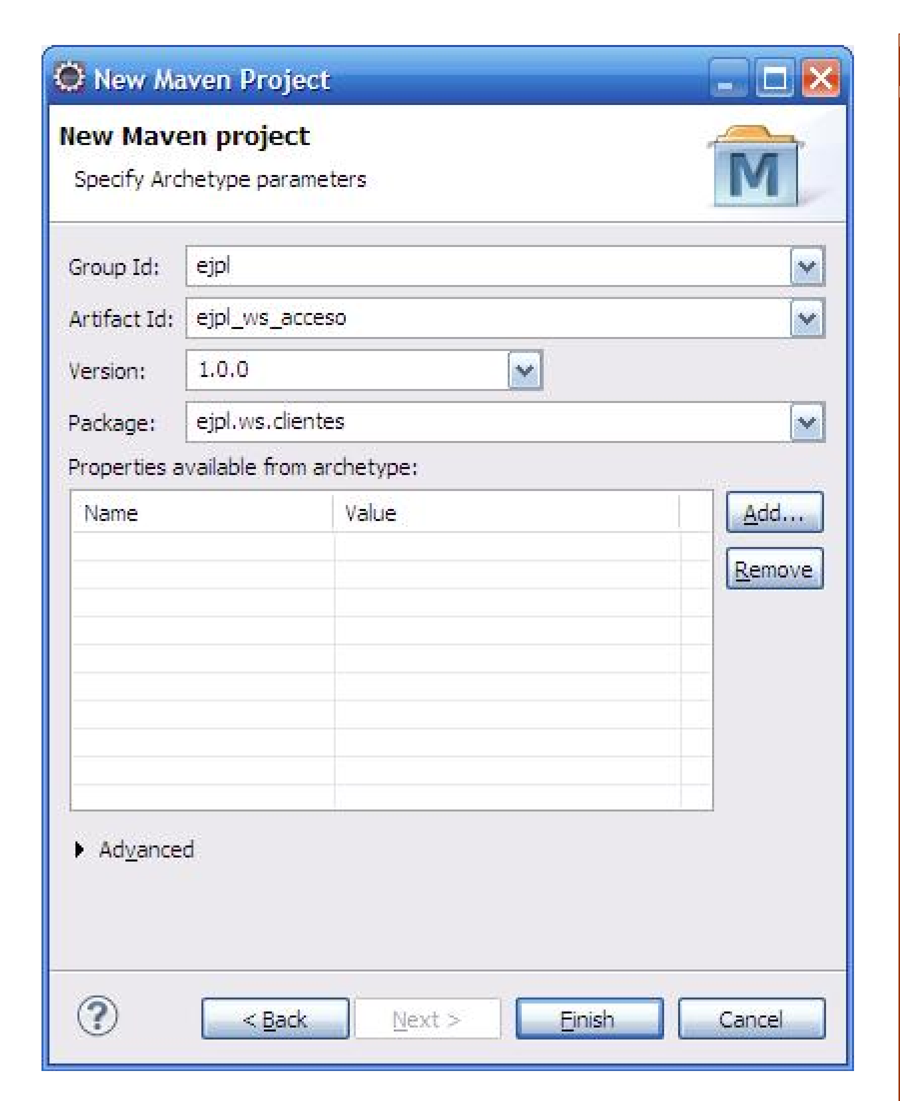
NOTA IMPORTANTE
Antes de generar un arquetipo es **IMPRESCINDIBLE** haber generado un arquetipo para albergar el proyecto (consultar "**ATLAS_MUS_Preparacion_Entorno_Desarrollo**" para ver cómo crear un **arquetipo de proyecto**). Cualquier arquetipo posterior será incluido como un módulo de éste.

Crear un proyecto de Servicios Web a partir de un arquetipo es similar a la creación de un proyecto Web, modificando el nombre del arquetipo a utilizar.

Para crear un proyecto a partir de un arquetipo maven puede consultar la guía paso a paso que se indica en el documento **ATLAS_MUS_Preparacion_Entorno_Desarrollo**, según se indica en el apartado "Creación de una aplicación Web desde cero", **sustituyendo "atlas-arquetipos-generador-web" por "atlas-arquetipos-generador-servicioweb"**.



A continuación se nos mostrará una pantalla para indicar cuáles son los parámetros con los que vamos a crear nuestro proyecto.



ATENCION – NOMENCLATURA
Es muy importante utilizar en esta pantalla los valores que cumplan la normativa de Atlas: **groupId**: Nombre del proyecto. Normalmente es de 4 caracteres y es un código que se le da a un proyecto. Todos los módulos de un proyecto tendrán el mismo *groupId* y se corresponderá con el nombre del proyecto. **artifactId**: Nombre del módulo. (En el caso de servicios web el nombre del módulo será: *xxxx_ws_yyyy* donde *xxxx* se corresponde con el *groupId* indicado y *yyyy* el texto que identifique a este modulo y lo diferencie de otros: Ejemplo: *ejpl_ws_acceso*. El carácter de separación a utilizar debe ser el guión bajo. **Version**: La primera versión será *1.0.0* **package**: El nombre del paquete java será el *groupId*, seguido de un punto y el nombre del bloque funcional dentro de la aplicación (para aplicaciones grandes se crearán varios bloques funcionales). Todos los nombres deben ir en minúsculas.

Al pulsar **Finish** se crea el nuevo proyecto.

ATENCION
Las clases que se incluyen de ejemplo son simplemente eso y deben ser eliminadas del proyecto cuando no se necesiten.

Antes de comenzar a visualizar la estructura del nuevo proyecto generado, debemos realizar un ajuste en el fichero ***pom.xml*** del nuevo proyecto creado. Para ello, debemos abrir el fichero "***web\pom.xml***", "***lib\pom.xml***" y "**test\pom.xml**", eliminar la sección ***<parent>*** existente y descomentar la sección ***<parent>*** que aparece comentada.

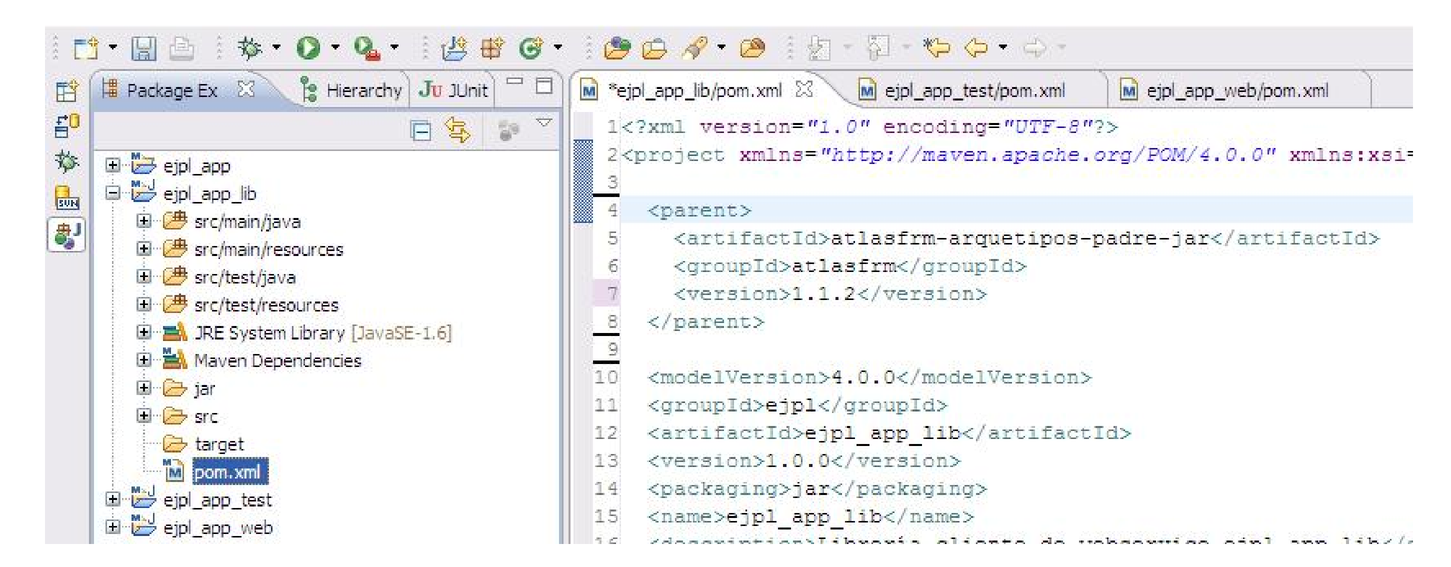
Antes de eliminar:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://ww

<!-- Descomentar estas líneas y eliminar la sección <parent> de abajo
  <parent>
    <artifactId>atlasfrm-arquetipos-padre-jar</artifactId>
    <groupId>atlasfrm</groupId>
    <version>1.1.2</version>
  </parent>
-->

  <parent>
    <artifactId>ejpl_app</artifactId>
    <groupId>ejpl</groupId>
    <version>1.0.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>ejpl</groupId>
  <artifactId>ejpl_app_lib</artifactId>
  <version>1.0.0</version>
```

Después de eliminar:



Una vez configurado el proyecto general y el subproyecto (módulo de tipo aplicación **servicioweb**), ya podemos seguir viendo la estructura del subproyecto para la aplicación que acabamos de generar.

2.2 Estructura del arquetipo

Este arquetipo genera un **proyecto maven multimodular** con un proyecto padre cuyo ***pom*** llevará la configuración general del resto de subproyectos. El proyecto consta de tres módulos:

- **Lib**: Módulo de tipo jar donde se implementa el cliente del servicio web. Aquí se crearán las interfaces de servicio y los objetos de datos de intercambio. Es además, una dependencia del proyecto Web ya que el webservice implementado tendrá necesariamente que hacer uso de los objetos definidos en este módulo.
- **Web**: Módulo de tipo war, configurado para utilizar Spring, Hibernate, Axis2 y seguridad, y preparado para desplegar el servicio web en un servidor de aplicaciones. Aquí se creará la implementación del servicio web.
- **Test**: Módulo con proyectos de SoapUI preparados para realizar pruebas de llamada sobre el servicio web. Contiene un proyecto SoapUI para cada uno de los ejemplos de servicio incluidos. Este módulo no generará ningún producto empaquetado como los otros dos módulos (paquete war el módulo web y paquete jar el módulo lib).

2.3 Estructura de directorios y archivos

Una vez generado el proyecto se generan una serie de directorios a estilo maven, donde ubicaremos los fuentes y recursos de las partes de java, test y aplicación.
A continuación se describe la estructura de cada uno de los dos módulos del proyecto.

2.3.1 Estructura del Módulo web

El módulo web tiene la siguiente estructura de directorios y ficheros:

	pom.xml	Fichero de configuración de Maven del proyecto
src/main/java	De esta ruta cuelga la estructura de paquetes con el **código fuente del proyecto**. El paquete raíz va a llamarse igual que el nombre del módulo que se está desarrollando. Inicialmente el paquete raíz se llamará con el código de proyecto (equivalente al **groupId**). Adicionalmente, en el subpaquete 'services' residirán las implementaciones de los servicios web a publicar. En los ejemplos inferiores se ha tomado como paquete raíz '**ejpl**'.	

	ejpl.webservice.services EjemploServiceImpl.java	Implementación del servicio web. Las interfaces del servicio web deben crearse en el módulo ***lib***.
src/main/resources	Ruta donde se generarán los recursos del proyecto, normalmente son ficheros de configuración y ficheros de propiedades. En nuestro caso contiene los siguientes ficheros:	
	environment.properties	Fichero de propiedades que contiene todas las **variables de configuración cuyo valor puede ser diferente en distintos entornos de ejecución** (local, desarrollo, producción, etc.). Antes de desplegar un proyecto en los entornos de ICM, este fichero será sustituido por su homólogo correspondiente al entorno en el que se vaya a desplegar.
	log4j.xml	Fichero de configuración de log4j; lleva además configuración de los módulos de trazas y monitorización.
conf	application.properties	Fichero de propiedades que contiene mediante clave y valor todos los valores particulares de la aplicación que no son susceptibles de ser modificados o configurados entre diferentes entornos de ejecución.
	applicationContext-dao.xml	Fichero de configuración de Spring para la carga en el contexto de Spring de todos los DAOs (clases de acceso a datos).
	applicationContext-database.xml	Fichero de configuración de Spring que lleva la configuración del ***dataSource***, ***sessionManager*** y ***transactionManager*** que integra Hibernate con Spring. Inicialmente el ***datasource*** será configurado mediante las propiedades ubicadas en el fichero **conf/jdbc.properties** (se generan de forma automática con maven) aunque para el paso a integración deben configurarse localizando por ***jndi*** al ***datasource*** del servidor de aplicaciones.
	applicationContext-general.xml	Fichero de configuración de Spring para la carga de las propiedades ubicadas en ***application.properties*** y resto de archivos de propiedades necesarios.
	applicationContext-services.xml	Fichero de configuración de Spring para la carga de todas las clases de Servicios.

src/test/java	Ruta donde se generarán el código fuente para los test de nuestro proyecto. Este código nunca deberá estar incluido en el jar final.	
	ejpl.webservice	Paquete para los tests unitarios. En el módulo web no hay tests implementados por defecto.
src/test/resources	Ruta donde se generarán los recursos para realizar las pruebas de test, normalmente son ficheros de configuración y ficheros de propiedades. En nuestro caso contiene los siguientes ficheros:	
	ssl keystore.keystore truststore.keystore	Ficheros de configuración por si se desea acceder a la aplicación por ssl
	conf applicationContext-database-test.xml	Lleva la configuración del dataSource, sessionManager y transactionManager que integra Hibernate con Spring. Este fichero es un espejo del fichero **conf/applicationContext-database.xml** excepto que en ningún caso debe localizar el datasource por JNDI del servidor de aplicaciones, puesto que en la fase de test no tenemos acceso a dicho servidor.
src main webapp	Ruta donde se generarán los recursos necesarios para la parte web de la aplicación.	
	index.html	Fichero de acceso principal que redirige al listado de servicios del web service.
	axis2-web listServices.jsp	Página de entrada que muestra un listado de servicios y un enlace a sus documentos WSDL.
	WEB-INF	services.xml Descriptor de despliegue de servicios de Axis2.
		web.xml Descriptor web de la aplicación
		weblogic.xml Descriptor adicional para el servidor de aplicaciones Weblogic.
		jetty-context.xml jetty-env.xml Descriptores adicionales para el servidor de aplicaciones Jetty.

generador	hibernate.cfg.xml hibernate.reveng.xml	Directorio para herramienta de generación automática de código.

2.3.2 Estructura del Módulo lib

El módulo lib tiene la siguiente estructura de directorios y ficheros:

lib	pom.xml	Fichero de configuración de maven del módulo ***lib***. Normalmente las dependencias no se incluyen en este fichero, sino en el fichero ***pom.xml*** del proyecto padre.
src/main/java	De esta ruta cuelga la estructura de paquetes con el **código fuente del proyecto cliente**. El paquete raíz va a llamarse igual que el nombre del módulo que se está desarrollando. Inicialmente el paquete raíz se llamará con el código de proyecto (equivalente al **groupId**). Adicionalmente, para cada bloque funcional de nuestra aplicación debe crearse un subpaquete.	
	ejpl.webservice.domain DatosEntrada.java DatosSalida.java	Entidades de entrada/salida del API del webservice
	ejpl.webservice.services EjemploService.java	Interfaces de los servicios web que serán publicados.
	ejpl.webservice.client ClienteEjemploService.java ClienteEjemploServiceWSS1.java ClienteEjemploServiceWSS2.java	Implementaciones de cliente de las interfaces de los servicios web.
src/main/resources	Ruta donde se generarán los recursos del proyecto, normalmente son ficheros de configuración y ficheros de propiedades. En nuestro caso contiene los siguientes ficheros:	
conf	applicationContext-ejpl_ws_lib.xml	Fichero de configuración de los servicios de cliente publicados.
META-INF	politicaWSSFirmado.xml politicaWSSFirmadoCifrado.xml	Ficheros de política de seguridad a aplicar si el servicio web es seguro
src/test/java	Ruta donde se generarán el código fuente para los test de nuestro proyecto. Este código nunca deberá estar incluido en el jar final.	

	ejpl.webservice.client ClienteEjemploServiceTest.java ClienteEjemploServiceWSS1Test.java ClienteEjemploServiceWSS2Test.java	Tests unitarios para comprobar la correcta implementación de los clientes del servicio web.
src/test/resources	Ruta donde se generarán los recursos para realizar las pruebas de test, normalmente son ficheros de configuración y ficheros de propiedades. En nuestro caso contiene los siguientes ficheros:	
	environment.properties	Fichero de propiedades que contiene mediante clave y valor todos los valores susceptibles de ser modificados o configurados entre diferentes entornos de ejecución. Este fichero solamente es para la ejecución de los test.
	log4j.xml	Fichero de configuración de log4j; lleva además configuración de los módulos de trazas y monitorización.
conf	applicationContext-test.xml	Contiene la configuración básica de un bean de Spring de ejemplo para ejecutar el test de JUnit que viene preconfigurado.
	application.properties	Fichero de propiedades que contiene mediante clave y valor todos los valores particulares de la aplicación que no son susceptibles de ser modificados o configurados entre diferentes entornos de ejecución.
jar adicional entregas formacion mantenimiento preproduccion produccion validacion	environment.properties	Directorios de uso interno de ICM para la configuración de la aplicación en los distintos entornos. Los ficheros environment.properties de estos directorios solamente se utilizan para la ejecución de los test (la librería es independiente del entorno). Si se incluye una nueva variable en src/main/resources/environment.properties se ha de incluir también en todos estos.

2.3.3 Estructura del Módulo test

El módulo test tiene la siguiente estructura de directorios y ficheros:

test	pom.xml	Fichero de configuración de maven del módulo ***test***. Normalmente las dependencias no se incluyen en este fichero, sino en el fichero ***pom.xml*** del proyecto padre.
src/main/java	De esta ruta cuelga la estructura de paquetes con el **código fuente**. En este módulo no se incluye código fuente. .	
src/main/resources	Ruta donde se generarán los recursos del proyecto. En nuestro caso contiene los siguientes ficheros:	
soapui	EjemploService-soapui-project.xml EjemploServiceWSS1-soapui-project.xml EjemploServiceWSS2-soapui-project.xml	Proyectos SoapUI para los servicios de ejemplo.
	client_ssl.jks client.jks	Almacenes de certificados para las pruebas con SoapUI.
wsdl	EjemploService.wsdl EjemploServiceWSS1.wsdl EjemploServiceWSS2.wsdl	Descriptores del ejemplo de servicio web.
src/test/java	Ruta donde se generarán el código fuente para los test de nuestro proyecto. Este módulo no tiene código java de tests unitarios.	
src/test/resources	Ruta donde se generarán los recursos para realizar las pruebas de test, normalmente son ficheros de configuración y ficheros de propiedades. Este módulo no tiene recursos para los tests java:	

2.4 Configuración del descriptor de servicios services.xml

Toda la configuración que se muestra a continuación se realiza según marca la documentación de Axis2, por lo que para cualquier consulta adicional puede consultarlo en la web http://axis.apache.org/axis2/java/core/docs/axis2config.html.

2.4.1 Configuración básica

El arquetipo genera el fichero *services.xml* en el módulo web con una configuración válida que despliega un bean del contexto de Spring como WebService. Si necesitamos editar dicho fichero lo podremos hacer de la siguiente manera:

- **service**: etiqueta que especifica un servicio, podemos publicar tantos servicios como deseemos siempre

que tengan un nombre diferente (parámetro *name*). Inicialmente se pone como nombre de servicio el nombre de la aplicación.

- **ServiceObjectSupplier**: especifica que clase nos proporciona los objetos que vamos a publicar, en nuestro caso Spring. Este parámetro no se debe modificar.
- **SpringBeanName**: Nombre del bean que vamos a publicar ubicado en el contexto de Spring (*applicationContext-services.xml*).
- **ServiceClass**: Interfaz de la clase que se va a publicar. Se debe especificar la interfaz del servicio que queremos publicar para que Axis2 sepa como generar el WSDL correspondiente, de lo contrario podemos tener problemas por que las clases estén interceptadas por Spring-AOP y esto falsee su interfaz en ejecución. Esta clase debe crearse en el módulo cliente ***lib*** aunque se referencie en este módulo.

A continuación se muestra un ejemplo de configuración del fichero *services.xml* del módulo ***web*** que publica un servicio.

Services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
    <service name="EjemploService">
        <parameter name="ServiceObjectSupplier">
org.apache.axis2.[...].SpringServletContextObjectSupplier</parameter>
        <parameter name="SpringBeanName">EjemploService</parameter>
        <parameter name="ServiceClass">ejpl.webservice.services.EjemploService</parameter>

        <messageReceivers>
            <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
                class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
            <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
                class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
        </messageReceivers>
    </service>

  <!-- añadir aquí mas servicios si se desea <service name="nuevoServicio"> ... </service> -->
</serviceGroup>
```

2.4.2 Configuración para seguridad

Para obtener información más detallada de la aplicación de seguridad en el servicio web, así como del contenido y operativa de los módulos, debe consultarse el documento ***ATLAS_MUS_Servicios_Web.doc***.

2.5 Despliegue y Ejecución de la aplicación

El arquetipo WebService es una aplicación web preparada para la publicación de alguno de sus servicios mapeados en Spring como servicio web Axis2.

Para construir el proyecto debe ejecutarse el siguiente comando desde el POM padre del proyecto generado:

mvn clean install

Por defecto, al ejecutar Maven con los parámetros "***clean install***" se genera el paquete ***jar*** correspondiente al cliente del servicio web y el ***war*** correspondiente a la aplicación web que contiene este. Para desplegar el ***war*** en nuestro servidor de aplicaciones podemos coger el fichero de la carpeta en la que se ha generado (**web/target/nombreFichero.war**), o podemos utilizar un servidor *Jetty* local que viene preconfigurado en el arquetipo. Para ejecutar directamente un servidor **jetty** con nuestro proyecto desplegado, debemos ejecutar la siguiente línea de comandos **desde el directorio web** de nuestro proyecto la siguiente línea de comandos:



```
mvn jetty:run
```

Esto arranca un servidor jetty en el puerto 9080 de nuestra máquina, en el que se encuentra desplegado el ***war*** de nuestra aplicación. Introduciendo la siguiente URL en un navegador se podrá ver un listado de los servicios desplegados. Esta URL solo es válida para el servidor Jetty local (la composición de esta URL para un servidor Weblogic se lista más abajo en este documento).

```
http://localhost:9080/services/listServices
```

En el caso de los ejemplos securidados el acceso es por

```
https://localhost:9443/services/listServices
```

Si en lugar de un servidor jetty queremos construir y desplegar en un servidor **WebLogic** definido por el usuario, es necesario primero configurar en el fichero ***environment.properties*** los parámetros del servidor a utilizar:

```
# Configuración para despliegue en weblogic local
weblogic.adminServer.hostName=localhost
weblogic.hostName=localhost
weblogic.adminServer.protocol=t3
weblogic.adminServer.port=7001
weblogic.port=7001
weblogic.servlet.port=7001
weblogic.user=weblogic
weblogic.password=contraseña
weblogic.serverName=NombreServidor
weblogic.targetNames=NombreTarget
weblogic.remote=false
weblogic.upload=false
```

Posteriormente puede ejecutarse la siguiente sentencia de maven, **desde el directorio web** del proyecto:

```
mvn clean install deploy –Pweblogic-local
```

Para la ejecución de los tests JMeter, desde el módulo test:


mvn clean install –Pjmeter

Para la generación y despliegue de los artefactos adicionales con los javadocs y el código fuente del proyecto:

mvn clean install –Dadditional_artifacts=true

Para probar la ejecución de este arquetipo puede comprobar en la URL donde se listan los servicios desplegados y sus documentos WSDL:

http://**<host>:<puerto>/<contexto>/**services/listServices

2.6 EJECUCION TEST

Para la ejecución de los test es necesario haber levantado previamente el servicio web en el servidor de aplicaciones que en local puede ser el propio jetty.

2.6.1 Ejecución de test unitarios

Los test unitarios se encuentran en el módulo **lib** en la carpeta src/test/java/xxxx/client. Antes de ejecutarlos es necesario editarlos y **quitar o comentar la anotación @Ignore** ya que de lo contrario no se van a ejecutar. Esta etiqueta se ha incluido por que si se compila ejecutando los test y no está levantado el servidor de aplicaciones con el servicio web lo test no van a funcionar.

Para comprobar la correcta implementación de la comunicación entre cliente y servidor, con el servidor del servicio web arrancado, se deberá ejecutar la siguiente sentencia **desde el directorio lib** del proyecto:

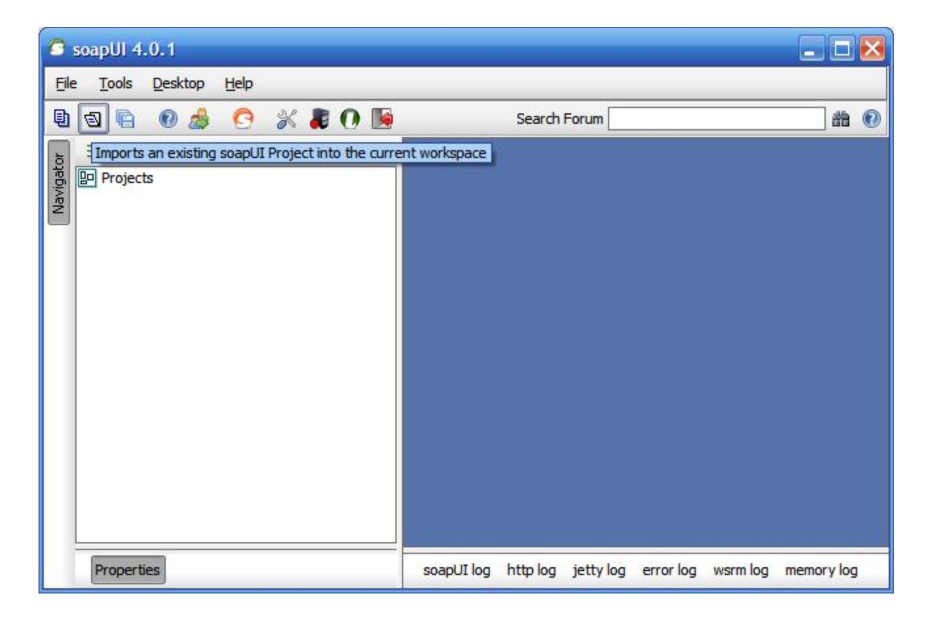
mvn test

También se pueden ejecutar cada test de Junit desde el propio Eclipse como se ejecuta cualquier clase de tipo Junit.

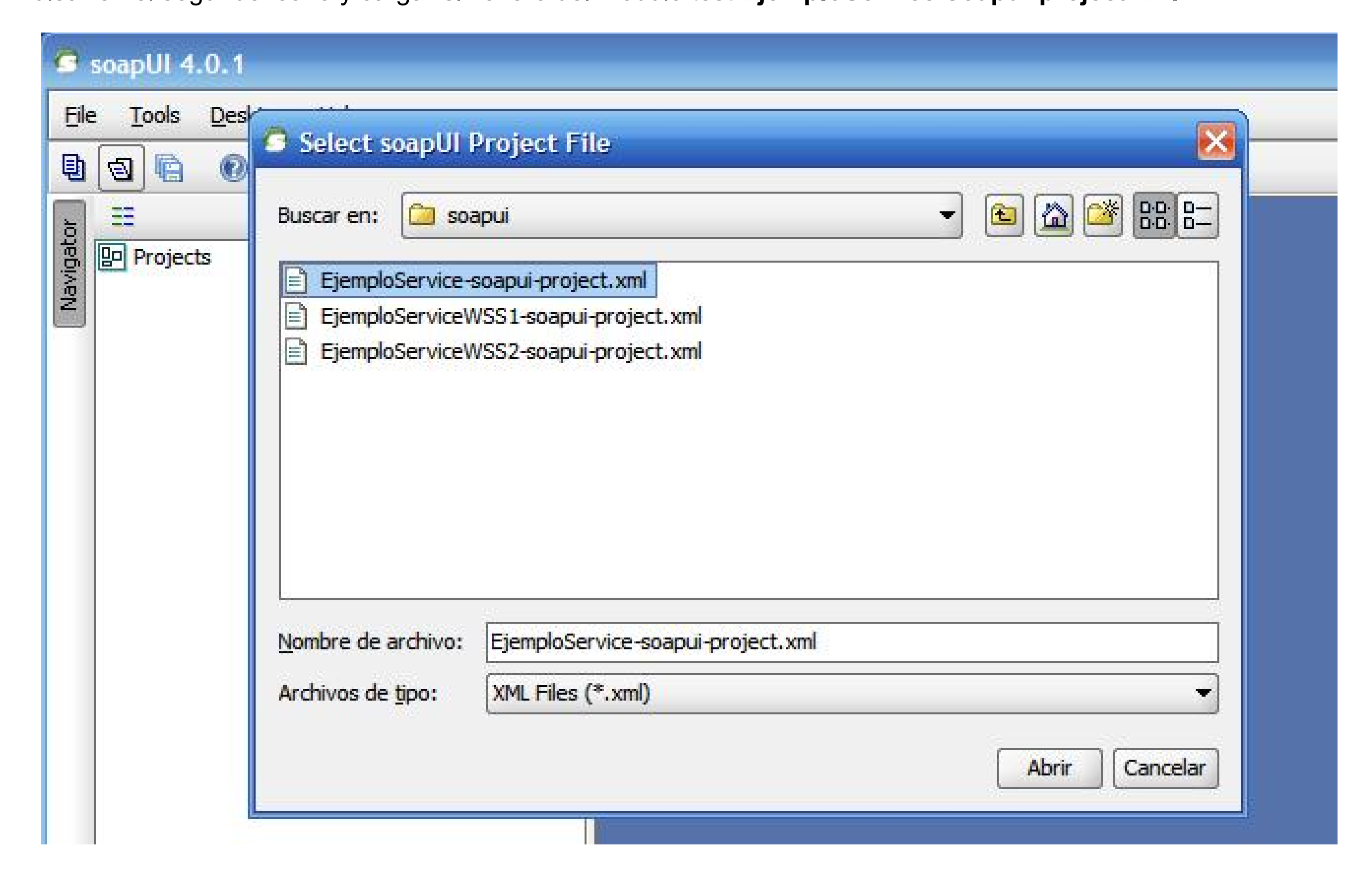
2.6.2 Ejecución de los tests de SoapUI

SoapUI es una aplicación para realizar llamadas a servicios web. Esta aplicación es utilizada normalmente para comprobar la correcta ejecución de los servicios web. Los proyectos de prueba incluidos en el módulo de test se han generado con la versión 4.0.1 de SoapUI. Esta aplicación puede ser descargada desde el siguiente enlace: http://sourceforge.net/projects/soapui/files/soapui/4.0.1/

Para realizar una prueba de llamada al servicio web, primero debe levantarse el servidor jetty tal y como se especifica en el apartado anterior. A continuación ejecutar SoapUI.



Pulsar en el segundo icono y cargar el fichero del módulo test **EjemploService-soapui-project.xml**.

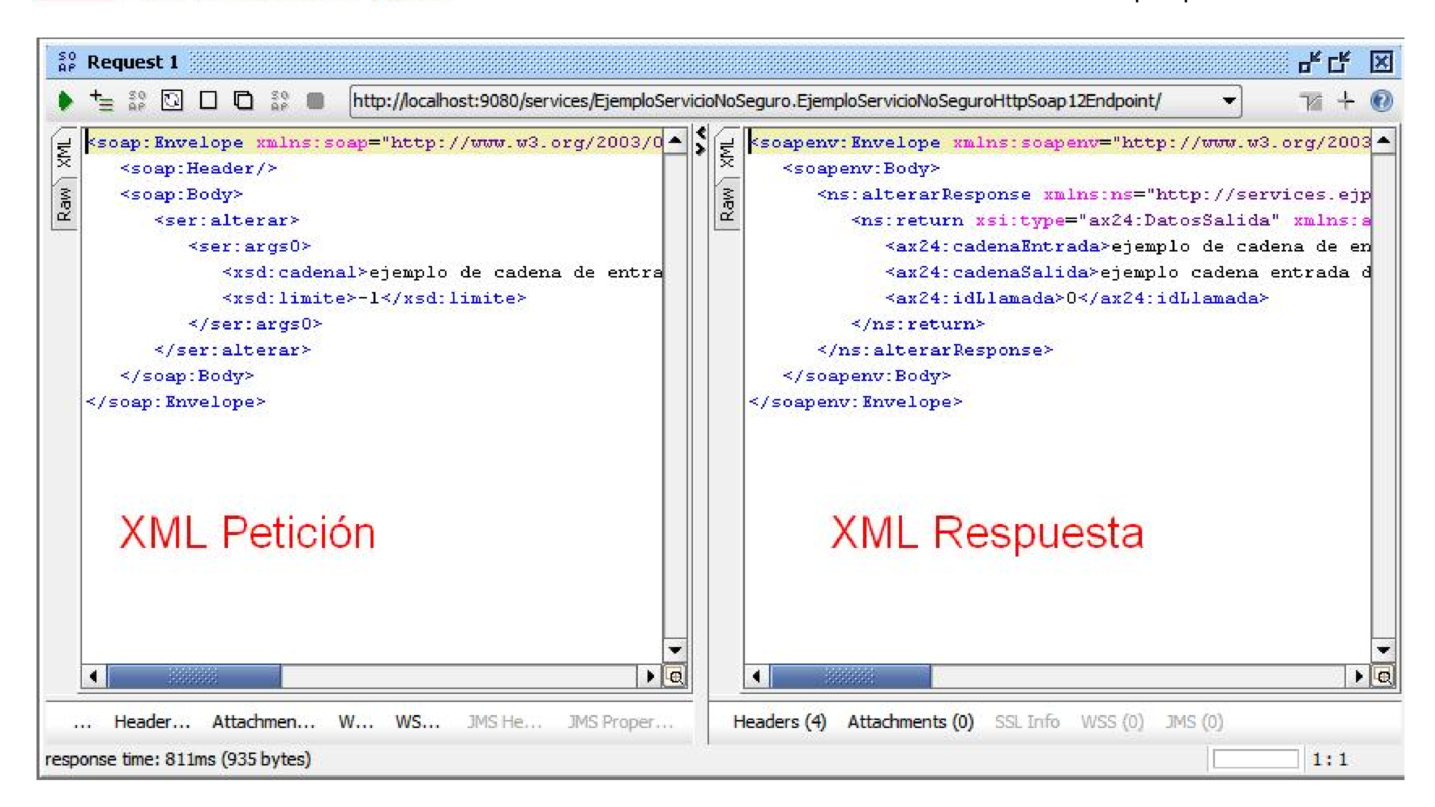


Una vez cargado el proyecto, desplegar el elemento **EjemploServiceSoap12Binding** (también puede probarse el

Soap11Binding) y el método **alterar**. Hacer doble click en **Request 1**. Con esto tendremos desplegado un XML de petición al servicio web.

```
<soap:Envelope xmlns:soap="http://w
   <soap:Header/>
   <soap:Body>
      <ser:alterar>
         <ser:args0>
            <xsd:cadenal>ejemplo de
            <xsd:limite>-1</xsd:lim
         </ser:args0>
      </ser:alterar>
   </soap:Body>
</soap:Envelope>
```

Para ejecutar esta petición hay que pulsar el icono ▶. Con esto se enviará la petición al servicio y se presentará la respuesta.



Para comprobar cualquier otro método de este proyecto o del proyecto de cliente seguro debe seguirse el mismo procedimiento.

3 VALIDACIÓN DE LA NORMATIVA Y GENERACIÓN DE LA DOCUMENTACIÓN

Los arquetipos vienen preparados para automáticamente generar un web-site con la información de la aplicación, incluyendo lava *javadoc*, información de dependencias, etc. También se ejecuta una herramienta automática que valida el cumplimiento de la normativa de Atlas, que genera un informe sobre los posibles incumplimientos de dicha normativa. Para más información sobre dicha herramienta consultar el documento **ATLAS_MUS_Preparacion_Entorno_Desarrollo**.

4 PREGUNTAS MAS FRECUENTES

La lista de preguntas frecuentes se encuentra en el portal de arquitectura.

5 ENLACES RELACIONADOS

Producto	URL
Axis2	http://ws.apache.org/axis2/
Configuración Services.xml	http://axis.apache.org/axis2/java/core/docs/axis2config.html
Apache Maven	http://maven.apache.org/
Log4J	http://logging.apache.org/log4j/
SoapUI	http://www.soapui.org/